Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Н[авчально-науковий інститут комп'ютерних наук та інформаційних технологій](https://web.kpi.kharkov.ua/if/uk/uaabout)

Кафедра «Системного аналізу та інформаційно-аналітичних технологій»

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

з дисципліни "Програмна технологія DOT NET"

про виконання Ідивідуального завдання

«Проект на мові програмування C# з використанням програмних технологій .NET для управління проектами та завданнями»

Виконав: студент групи КН-1022в

Бєдарєва Г.А.

Перевірив: доцент кафедри САІТ

Івашко А.В.

Харків-2024

ЗМІСТ

Вступ…………………………………………………………………………….... 3

1 План розробки додатку………………………………………………………….7

2 Проектування та розробка бази даних…………..……………………………10

3 Проектування та реалізація класів…………………………………………... 15 4 Розробка інтерфейсу та реалізація реєстрації і входу………………………..32

5 Розробка основної сторінки з проектами і таймером………………………..37

6 Розробка форми для додавання проектів……………………………………..44

7 Розробка форми управління завданнями……………………………………..47

8 Розробка форми для перегляду статистики…………………………………..51

9.Висновки та перспективи розвитку….…….…………………….……………55

**Вступ**

Сучасний ритм життя та стрімкий розвиток технологій ставлять перед суспільством завдання організації ефективного управління часом і ресурсами. Зокрема, особливо важливим є планування та контроль проєктів у різних галузях, таких як IT, освіта, будівництво чи особистий розвиток. Для цього створюються програмні продукти, що дозволяють користувачам організовувати свої завдання, відстежувати прогрес виконання проєктів, а також оптимізувати використання часу. Мій проєкт Project Tracker реалізує ці функції та пропонує простий, але функціональний інструмент для управління проєктами.

Актуальність проєкту обумовлена кількома факторами:

1. Потреба в управлінні проєктами: Сьогодні кожна організація, незалежно від масштабу, зіштовхується з необхідністю контролювати свої завдання та ресурси. Project Tracker спрощує цю задачу для окремих користувачів або невеликих груп.
2. Відсутність надмірно складних інструментів: Багато існуючих рішень, таких як Microsoft Project або Jira, є надто складними для звичайного користувача. Проєкт Project Tracker орієнтований на простоту використання, зберігаючи ключові функції.
3. Популярність C# та WinForms: Використання C# як мови програмування та WinForms як інструмента розробки обумовлено їх надійністю, доступністю та широкими можливостями для створення додатків із графічним інтерфейсом.

***Чому реалізація на C#?***

Я обрала мову C# і платформу .NET для реалізації даного проєкту через їх переваги:

1. Простота та зручність розробки: C# має інтуїтивно зрозумілий синтаксис, який дозволяє швидко створювати додатки, орієнтовані на кінцевого користувача. Завдяки платформі .NET, розробка є ефективною, оскільки надається велика кількість бібліотек і засобів для роботи з графічним інтерфейсом, базами даних та іншими компонентами.
2. Інтеграція з базами даних: Проєкт Tracker вимагає надійного збереження даних про користувачів, проєкти, завдання та час виконання. Завдяки технологіям, таким як ADO.NET чи Entity Framework, C# забезпечує гнучкий інструментарій для роботи з реляційними базами даних.
3. Графічний інтерфейс: Для створення зручного інтерфейсу користувача було використано Windows Forms. Це дозволяє забезпечити візуальну привабливість додатка, зберігаючи при цьому простоту у використанні.
4. Масштабованість: Використання C# дає можливість у майбутньому масштабувати програму — інтегрувати додаткові модулі, наприклад, підтримку хмарного збереження даних чи аналітичні функції.

***Наукові задачі проєкту***

У рамках реалізації даного проєкту поставлено кілька наукових та технічних задач:

1. Створення ефективної структури збереження даних. Завданням є розробка реляційної бази даних для зберігання проєктів, користувачів, завдань та записів часу. Для цього потрібно забезпечити оптимізацію доступу до даних та мінімізацію надлишковості через нормалізацію структури бази даних.
2. Розробка інтуїтивного інтерфейсу користувача. Мета — створити зручний та зрозумілий графічний інтерфейс, який дозволить користувачам швидко додавати, редагувати проєкти, відстежувати час та завершувати завдання. Це включає реалізацію таких елементів, як кнопки, форми, випадаючі списки.
3. Реалізація функціоналу відстеження часу. Для кожного завдання користувач може запускати таймер, який обраховує витрачений час, і записувати цей показник у базу даних. Виклик полягає в синхронізації таймера з даними у базі.
4. Безпека даних. Забезпечення захисту даних через механізми авторизації та аутентифікації користувачів. Проєкт реалізує можливість реєстрації та входу з перевіркою введених даних.

***Методи виконання задач***

* Аналіз та проєктування: На першому етапі було виконано аналіз вимог до додатка. Це включало вивчення потреб потенційних користувачів та вибір найефективніших засобів реалізації.
* Розробка інтерфейсу: Використання Windows Forms дозволяє швидко створювати інтуїтивно зрозумілі форми для взаємодії користувача з програмою.
* Розробка структури бази даних: Виконано проєктування бази даних за допомогою інструментів, таких як HeidiSQL, що забезпечує ефективність зберігання даних.
* Кодування: Використовуючи C#, було реалізовано всю функціональність, зокрема роботу з базою даних, облік часу та управління проєктами.
* Тестування: Виконувалося ретельне тестування додатка, щоб перевірити його стабільність, зручність і коректність роботи.

Проєкт Project Tracker — це інноваційний та зручний інструмент для управління проєктами, який дозволяє користувачам зберігати, контролювати та оптимізувати свої завдання. Використання C# забезпечує надійність роботи програми, а Windows Forms — інтуїтивно зрозумілий інтерфейс. Реалізація цього проєкту вирішує низку актуальних задач, пов’язаних з управлінням проєктами, і має великий потенціал для подальшого розвитку.

У пояснювальній записці детально описано весь процес розробки додатку **Project Tracker** – від початкового планування до аналізу результатів і досягнення поставлених цілей. Особливу увагу приділено проєктуванню архітектури програми, розробці функціоналу, а також інтеграції бази даних для збереження даних про проєкти, завдання та користувачів. Відображено ключові етапи роботи: аналіз вимог, реалізацію інтерфейсу, розробку алгоритмів і тестування додатку. У тексті наведено технічні деталі, приклади коду та пояснення логіки роботи основних компонентів. Завершальною частиною є аналіз ефективності розробленого рішення, а також рекомендації щодо подальшого вдосконалення додатку, таких як інтеграція нових функцій і покращення продуктивності. Цей документ надає повне уявлення про процес створення додатку та слугує основою для його подальшого розвитку.

**1. План розробки додатку**

***Аналіз вимог до додатку***

Першим етапом розробки додатку Project Tracker стало визначення його функціональних можливостей, орієнтованих на зручність та ефективність управління проектами. Додаток повинен задовольняти наступні вимоги:

* Реєстрація користувачів: Користувачі мають мати можливість створювати облікові записи, вказуючи унікальне ім’я користувача, електронну пошту та надійний пароль. Реєстрація забезпечує персоналізацію доступу до даних.
* Авторизація користувачів: Після успішної реєстрації користувачі повинні входити в систему, щоб отримати доступ до свого персонального робочого простору.
* Управління проектами: Основною функцією додатку є створення проектів. Користувачі повинні мати змогу створювати нові проекти, редагувати їхні деталі та видаляти ті, що вже не потрібні.
* Додавання завдань до проектів: Кожен проект може містити список завдань. Користувачі можуть додавати нові завдання, а також видаляти ті, що стали зайвими.
* Відстеження часу: Таймер для кожного проекту дозволяє користувачам фіксувати витрачений час. Цей час автоматично додається до загального часу виконання проекту.
* Перегляд статистики: Додаток повинен забезпечувати можливість перегляду загальної активності користувача. Це включає дані про всі проекти, загальний час роботи, а також детальну статистику по кожному проекту.

Ці вимоги були сформульовані з метою створення функціонального та зручного інструменту для управління проектами.

***Розробка бази даних***

На наступному етапі було створено структуру бази даних, яка забезпечує надійне зберігання даних користувачів, проектів, завдань і витраченого часу. Ключові таблиці бази даних:

* Users: для зберігання інформації про зареєстрованих користувачів.
* Projects: для зберігання деталей проектів, включаючи назву, статус та загальний час виконання.
* Tasks: для зберігання списків завдань, пов’язаних із проектами.
* TimeEntries: для фіксування часових записів, які відображають активність користувача над проектами.

Особливу увагу приділено оптимізації таблиць, забезпеченню зв’язків між ними за допомогою первинних і зовнішніх ключів. Це дозволило зробити базу даних масштабованою і легкою для обслуговування.

***Розробка необхідних класів***

Для побудови логіки додатку були спроектовані та реалізовані такі класи:

* User – відповідає за управління інформацією про користувачів, включаючи авторизацію та перевірку доступу.
* Project – реалізує функціонал створення, редагування, видалення проектів, а також обробку їхнього статусу.
* Task – опрацьовує завдання, що додаються до проектів, дозволяє їх створювати та видаляти.
* Timer – відповідає за запуск, зупинку та фіксацію витраченого часу.
* Statistics – надає користувачеві можливість переглядати загальну аналітику роботи над проектами.

Ці класи створюють основу для бізнес-логіки додатку.

***Розробка інтерфейсу та реалізація реєстрації і входу***

Для інтерактивної взаємодії з користувачем було створено зручний графічний інтерфейс на основі WinForms. Форми реєстрації та входу забезпечують перевірку введених даних, а також відповідають сучасним вимогам до дизайну. Реалізація включає інтеграцію з базою даних для перевірки користувачів.

***Основна сторінка з проектами і таймером***

Ця сторінка є центральною у додатку. Вона надає можливість управління проектами:

* створення, редагування та видалення проектів,
* запуск та зупинку таймерів,
* додавання та видалення завдань.  
  Користувачі можуть візуально відстежувати свій прогрес завдяки інтеграції таймерів із проектами.

***Вікно для відображення статистики***

На завершальному етапі було створено окреме вікно для перегляду статистики активності. Тут користувачі можуть бачити загальну кількість витраченого часу на проекти, аналізувати продуктивність і переглядати дані в структурованій формі.

Цей план охоплює всі етапи розробки додатку, забезпечуючи системний підхід до реалізації поставлених завдань.

**2. Проектування та розробка бази даних**

Розробка бази даних для проєкту Project Tracker є ключовим етапом, оскільки саме вона забезпечує зберігання, управління та обробку інформації, необхідної для функціонування додатку. База даних складається з чотирьох взаємопов’язаних таблиць: users, projects, tasks та timeentries, кожна з яких виконує конкретну роль у системі.

***1. Визначення структури бази даних***

На етапі проєктування було враховано всі основні вимоги додатку. Структура бази даних побудована з урахуванням нормалізації даних та забезпечення зв’язків між таблицями через зовнішні ключі. Це дозволяє уникнути надлишковості даних і спрощує їх подальшу обробку.

***2. Таблиця* users**

Таблиця users містить інформацію про всіх зареєстрованих користувачів додатку. Її поля:

* UserID — унікальний ідентифікатор користувача, який є первинним ключем (PRIMARY KEY). Його тип даних — INT, він автоматично збільшується при додаванні нового запису (AUTO\_INCREMENT).
* Username — ім’я користувача, типу VARCHAR(50). Це поле має бути унікальним (UNIQUE KEY), щоб запобігти дублюванню імен.
* Email — електронна адреса користувача, типу VARCHAR(100). Це поле також є унікальним.
* PasswordHash — хеш пароля, що забезпечує безпеку даних користувачів. Тип даних — VARCHAR(250).

Код створення таблиці:

CREATE TABLE `users` (

`UserID` int NOT NULL AUTO\_INCREMENT,

`Username` varchar(50) NOT NULL,

`Email` varchar(100) NOT NULL,

`PasswordHash` varchar(250) NOT NULL,

PRIMARY KEY (`UserID`),

UNIQUE KEY `Username` (`Username`),

UNIQUE KEY `Email` (`Email`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

***3. Таблиця* projects**

Таблиця projects відповідає за зберігання інформації про проекти, які створюють користувачі. Її поля:

* ProjectID — унікальний ідентифікатор проекту (PRIMARY KEY), типу INT, AUTO\_INCREMENT.
* UserId — зовнішній ключ (FOREIGN KEY), який пов’язує проєкт із користувачем. Це поле посилається на UserID у таблиці users.
* ProjectName — назва проєкту, типу VARCHAR(100).
* TotalTime — загальний час, витрачений на проєкт (типу INT), який обчислюється на основі часових записів.

Код створення таблиці:

CREATE TABLE `projects` (

`ProjectID` int NOT NULL AUTO\_INCREMENT,

`UserId` int DEFAULT NULL,

`ProjectName` varchar(100) NOT NULL,

`TotalTime` int DEFAULT ‘0’,

PRIMARY KEY (`ProjectID`),

KEY `UserId` (`UserId`),

CONSTRAINT `projects\_ibfk\_1` FOREIGN KEY (`UserId`) REFERENCES `users` (`UserID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

Зв’язок між projects та users забезпечує, що кожен проєкт належить певному користувачеві.

***4. Таблиця* tasks**

Таблиця tasks використовується для зберігання завдань, які створюються в межах кожного проєкту. Її поля:

* TaskID — унікальний ідентифікатор завдання (PRIMARY KEY), типу INT, AUTO\_INCREMENT.
* ProjectID — зовнішній ключ (FOREIGN KEY), який пов’язує завдання з конкретним проєктом. Це поле посилається на ProjectID у таблиці projects.
* TaskName — назва завдання, типу VARCHAR(100).

Особливістю таблиці є те, що завдання автоматично видаляються при видаленні відповідного проєкту завдяки опції ON DELETE CASCADE.

Код створення таблиці:

CREATE TABLE `tasks` (

`TaskID` int NOT NULL AUTO\_INCREMENT,

`ProjectID` int DEFAULT NULL,

`TaskName` varchar(100) NOT NULL,

PRIMARY KEY (`TaskID`),

KEY `tasks\_ibfk\_1` (`ProjectID`),

CONSTRAINT `tasks\_ibfk\_1` FOREIGN KEY (`ProjectID`) REFERENCES `projects` (`ProjectID`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

***5. Таблиця* timeentries**

Таблиця timeentries зберігає інформацію про часові записи для кожного проєкту. Поля:

* TimeEntryID — унікальний ідентифікатор запису (PRIMARY KEY), типу INT, AUTO\_INCREMENT.
* UserID — зовнішній ключ (FOREIGN KEY), що посилається на UserID у таблиці users.
* StartTime, EndTime — час початку та завершення роботи над проєктом, типу DATETIME.
* Duration — тривалість у хвилинах, типу INT.
* ProjectID — зовнішній ключ (FOREIGN KEY), що посилається на ProjectID у таблиці projects.
* EntryDate — дата запису, типу DATE.

Код створення таблиці:

CREATE TABLE `timeentries` (

`TimeEntryID` int NOT NULL AUTO\_INCREMENT,

`UserID` int DEFAULT NULL,

`StartTime` datetime NOT NULL,

`EndTime` datetime NOT NULL,

`Duration` int DEFAULT NULL,

`ProjectID` int DEFAULT NULL,

`EntryDate` date NOT NULL,

PRIMARY KEY (`TimeEntryID`),

KEY `UserID` (`UserID`),

KEY `ProjectID` (`ProjectID`),

CONSTRAINT `timeentries\_ibfk\_2` FOREIGN KEY (`UserID`) REFERENCES `users` (`UserID`),

CONSTRAINT `timeentries\_ibfk\_3` FOREIGN KEY (`ProjectID`) REFERENCES `projects` (`ProjectID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

***6. Заповнення бази даних***

База даних спочатку порожня і заповнюватиметься даними через інтерфейс користувача додатку. Реєстрація користувачів, створення проєктів, додавання завдань і запис часових даних виконуватимуться за допомогою відповідного функціоналу додатку.

***7. Перевірка коректності***

Після розробки бази даних тестуються всі CRUD-операції (створення, читання, оновлення, видалення). Зокрема, перевіряється:

* Додавання нових користувачів, проєктів, завдань і часових записів.
* Коректність зв’язків між таблицями.
* Правильна робота обмежень і залежностей (наприклад, видалення проєкту автоматично видаляє всі пов’язані завдання).

***8. Візуалізація схеми***

Схема бази даних (рис.2.1) демонструє всі зв’язки між таблицями та використані ключі. Це дозволяє візуалізувати структуру та оцінити логічну цілісність розробленої бази даних.

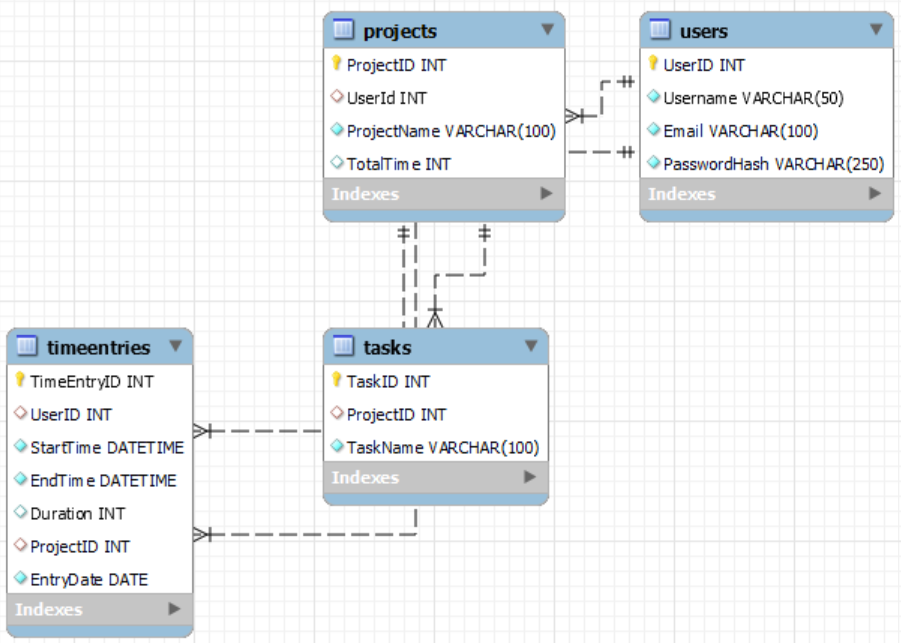


Рисунок 2.1 – Структурна схема бази даних

**3. Проектування та реалізація класів**

***3.1.Опис розробки класу User***

Клас User розроблений для моделювання сутності користувача в додатку «Project Tracker» та забезпечення управління даними користувачів. Він надає необхідну функціональність для створення, аутентифікації, пошуку та оновлення даних користувачів у базі даних.

***Основні властивості класу***

1. UserID: Унікальний ідентифікатор користувача, автоматично створюється базою даних. Є доступним лише для читання після створення користувача.
2. Username: Ім’я користувача, яке є унікальним і використовується для ідентифікації в системі.
3. Email: Електронна адреса користувача, що використовується для входу в систему.
4. PasswordHash: Хеш пароля користувача, що забезпечує безпечне зберігання паролів.

***Конструктори:***

1. User(string username, string email, string passwordHash): Використовується для створення нового користувача, коли його ID ще не створено.
2. User(int userId, string username, string email, string passwordHash): Використовується для ініціалізації об’єкта з даними, отриманими з бази даних, включаючи унікальний UserID.

***Методи:***

1. Save():  
   Цей метод використовується для збереження нового користувача в базі даних. Він:
   * Відкриває з’єднання з базою даних.
   * Виконує SQL-запит на вставку нового запису до таблиці Users.
   * Після успішного виконання запиту отримує UserID для новоствореного запису.
2. Authenticate(string email, string passwordHash) *(статичний метод)*:  
   Дозволяє аутентифікувати користувача на основі його електронної пошти та хешу пароля. Метод:
   * Шукає запис у базі даних з відповідними Email та PasswordHash.
   * Повертає об’єкт User, якщо користувач знайдений, або null, якщо дані невірні.
3. FindByEmail(string email) *(статичний метод)*:  
   Використовується для пошуку користувача за його електронною адресою. Метод:
   * Повертає об’єкт User, якщо відповідний запис знайдено в базі даних.
   * Повертає null, якщо користувача не знайдено.
4. UpdatePassword(string newPasswordHash):  
   Оновлює хеш пароля для існуючого користувача. Метод:
   * Приймає новий хеш пароля.
   * Виконує SQL-запит для оновлення поля PasswordHash у базі даних для відповідного UserID.

***Код класу* User:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using MySql.Data.MySqlClient;

namespace ProjectTracker

{

public class User

{

public int UserID { get; private set; }

public string Username { get; set; }

public string Email { get; set; }

public string PasswordHash { get; set; }

public User(string username, string email, string passwordHash)

{

Username = username;

Email = email;

PasswordHash = passwordHash;

}

public User(int userId, string username, string email, string passwordHash)

{

UserID = userId;

Username = username;

Email = email;

PasswordHash = passwordHash;

}

public void Save()

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «INSERT INTO Users (Username, Email, PasswordHash) VALUES (@Username, @Email, @PasswordHash)»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@Username», Username);

cmd.Parameters.AddWithValue(«@Email», Email);

cmd.Parameters.AddWithValue(«@PasswordHash», PasswordHash);

cmd.ExecuteNonQuery();

UserID = (int)cmd.LastInsertedId;

}

}

}

public static User Authenticate(string email, string passwordHash)

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «SELECT \* FROM Users WHERE Email = @Email AND PasswordHash = @PasswordHash»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@Email», email);

cmd.Parameters.AddWithValue(«@PasswordHash», passwordHash);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new User(

reader.GetInt32(«UserID»),

reader.GetString(«Username»),

reader.GetString(«Email»),

reader.GetString(«PasswordHash»)

);

}

else

{

return null;

}

}

}

}

}

public static User FindByEmail(string email)

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «SELECT \* FROM Users WHERE Email = @Email»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@Email», email);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

return new User(

reader.GetInt32(«UserID»),

reader.GetString(«Username»),

reader.GetString(«Email»),

reader.GetString(«PasswordHash»)

);

}

else

{

return null;

}

}

}

}

}

public void UpdatePassword(string newPasswordHash)

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «UPDATE Users SET PasswordHash = @PasswordHash WHERE UserID = @UserID»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@PasswordHash», newPasswordHash);

cmd.Parameters.AddWithValue(«@UserID», UserID);

cmd.ExecuteNonQuery();

}

}

}

}

}

***Призначення та використання***

Клас User виконує ключову роль у додатку, оскільки реалізує базові операції управління користувачами, такі як реєстрація, аутентифікація та оновлення даних. Він інтегрується з базою даних MySQL через бібліотеку MySql.Data.MySqlClient і забезпечує чіткий інтерфейс для роботи з користувачами, мінімізуючи прямий доступ до SQL-запитів.

Детальна обробка даних у класі сприяє дотриманню принципів безпеки, таких як використання хешування паролів, що гарантує захист конфіденційних даних користувачів.

***3.2.Опис розробки класу Project***

Клас Project створений для моделювання сутності проекту в додатку «Project Tracker» та забезпечення основного функціоналу управління проектами користувачів. Цей клас підтримує роботу з базою даних MySQL, дозволяючи створювати, редагувати, видаляти та отримувати інформацію про проекти разом із їхніми завданнями.

### ***Основні властивості класу:***

1. ProjectID: Унікальний ідентифікатор проекту, автоматично генерується базою даних після створення.
2. UserID: Ідентифікатор користувача, якому належить проект. Використовується для встановлення зв’язку між користувачами та їхніми проектами.
3. ProjectName: Назва проекту, задається користувачем під час створення.
4. TotalTime: Загальний час, витрачений на проект. Оновлюється при запуску та зупинці таймера.
5. Tasks: Колекція завдань, пов’язаних із проектом. Реалізована як список об’єктів класу Task.

### ***Конструктори:***

1. Project(int userId, string projectName): Використовується для створення нового проекту, коли його ProjectID ще не задано. Ініціалізує UserID, ProjectName та створює порожній список завдань.

### ***Методи:***

1. Save():  
   Додає новий запис про проект у базу даних. Метод:
   * Відкриває з’єднання з базою даних.
   * Виконує SQL-запит для збереження значень UserID, ProjectName, TotalTime.
   * Отримує та зберігає згенерований ProjectID.
2. AddTask(Task task):

Додає нове завдання до проекту. Метод:

* + Додає об’єкт завдання до списку Tasks.
  + Викликає метод Save класу Task, щоб зберегти завдання в базі даних із прив’язкою до поточного проекту.

1. Delete():  
   Видаляє проект із бази даних. Метод:
   * Виконує SQL-запит для видалення запису проекту за його ProjectID.
2. GetProjectsByUser(int userId) (статичний метод):

Повертає список всіх проектів, створених конкретним користувачем. Метод:

* + Виконує SQL-запит для отримання всіх записів із таблиці Projects, що відповідають UserID.
  + Для кожного проекту завантажує його завдання через метод LoadTasks.

1. LoadTasks():  
   Завантажує всі завдання, пов’язані з проектом, і додає їх до списку Tasks. Метод:
   * Виконує SQL-запит до таблиці Tasks для отримання всіх завдань, що відповідають ProjectID.
   * Додає кожне завдання до списку Tasks.
2. UpdateTotalTime(int additionalTime):

Оновлює загальний час, витрачений на проект. Метод:

* + Додає до TotalTime значення additionalTime.
  + Виконує SQL-запит для оновлення поля TotalTime у базі даних.

### Функціональність:

Клас Project забезпечує повний набір операцій для управління проектами:

* Додавання нових проектів.
* Завантаження проектів користувача разом із їхніми завданнями.
* Додавання завдань до проекту.
* Видалення проектів.
* Оновлення часу, витраченого на проект.

Клас інтегрується з класом Task для роботи із завданнями, що дозволяє підтримувати ієрархічну структуру даних: один проект може містити кілька завдань. Реалізація методів із використанням бібліотеки MySql.Data.MySqlClient забезпечує прямий доступ до бази даних для виконання необхідних операцій.

***Код класу* Project:**

using System.Collections.Generic;

using MySql.Data.MySqlClient;

namespace ProjectTracker

{

public class Project

{

public int ProjectID { get; private set; }

public int UserID { get; private set; }

public string ProjectName { get; set; }

public int TotalTime { get; set; }

public List<Task> Tasks { get; private set; }

public Project(int userId, string projectName)

{

UserID = userId;

ProjectName = projectName;

Tasks = new List<Task>();

}

public void Save()

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «INSERT INTO Projects (UserID, ProjectName, TotalTime) VALUES (@UserID, @ProjectName, @TotalTime)»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@UserID», UserID);

cmd.Parameters.AddWithValue(«@ProjectName», ProjectName);

cmd.Parameters.AddWithValue(«@TotalTime», TotalTime);

cmd.ExecuteNonQuery();

ProjectID = (int)cmd.LastInsertedId;

}

}

}

public void AddTask(Task task)

{

Tasks.Add(task);

task.Save(ProjectID);

}

public void Delete()

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «DELETE FROM Projects WHERE ProjectID = @ProjectID»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@ProjectID», ProjectID);

cmd.ExecuteNonQuery();

}

}

}

public static List<Project> GetProjectsByUser(int userId)

{

List<Project> projects = new List<Project>();

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «SELECT \* FROM Projects WHERE UserID = @UserID»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@UserID», userId);

using (var reader = cmd.ExecuteReader())

{

while (reader.Read())

{

Project = new Project(userId, reader.GetString(«ProjectName»))

{

ProjectID = reader.GetInt32(«ProjectID»),

TotalTime = reader.GetInt32(«TotalTime»)

};

project.LoadTasks();

projects.Add(project);

}

}

}

}

return projects;

}

public void LoadTasks()

{

Tasks.Clear();

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «SELECT \* FROM Tasks WHERE ProjectID = @ProjectID»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@ProjectID», ProjectID);

using (var reader = cmd.ExecuteReader())

{

while (reader.Read())

{

Task task = new Task(reader.GetString(«TaskName»))

{

TaskID = reader.GetInt32(«TaskID»),

};

Tasks.Add(task);

}

}

}

}

}

public void UpdateTotalTime(int additionalTime)

{

TotalTime += additionalTime;

using (var connection = new Database().GetConnection())

{

connection.Open();

string updateQuery = «UPDATE Projects SET TotalTime = @TotalTime WHERE ProjectID = @ProjectID»;

using (var cmd = new MySqlCommand(updateQuery, connection))

{

cmd.Parameters.AddWithValue(«@TotalTime», TotalTime);

cmd.Parameters.AddWithValue(«@ProjectID», ProjectID);

cmd.ExecuteNonQuery();

}

}

}

}

}

Таким чином, клас Project виконує важливу роль у додатку, надаючи користувачам зручний інструмент для управління проектами та контролю за виконанням завдань.

***3.3.Опис розробки класу Task***

Клас Task є частиною моделі додатку Project Tracker, яка забезпечує управління завданнями для проектів. Цей клас дозволяє створювати, зберігати, видаляти завдання та отримувати список завдань для конкретного проекту. Його інтеграція з базою даних MySQL реалізована через бібліотеку MySql.Data.MySqlClient, що забезпечує стабільну та ефективну взаємодію з таблицею Tasks.

***Основні властивості класу***

* TaskID *(тип: int)*: Ідентифікатор завдання, автоматично генерується базою даних після збереження завдання. Властивість доступна лише для читання, оскільки її значення визначається зовнішньою системою.
* TaskName *(тип: string)*: Назва завдання, яка задається користувачем при його створенні.

***Основний функціонал класу***

1. Конструктор
   * Task(string taskName)

Конструктор дозволяє створювати об’єкти класу Task, вказуючи лише ім’я завдання. Усі інші властивості задаються автоматично або через методи класу.

1. Метод Save

public void Save(int projectId)

Метод використовується для збереження нового завдання в базу даних. Він:

* + Відкриває з’єднання з базою даних.
  + Виконує SQL-запит для додавання запису в таблицю Tasks із зазначеними значеннями ProjectID та TaskName.
  + Зберігає ідентифікатор нового завдання в полі TaskID.

1. Метод Delete

public void Delete()

Видаляє завдання з бази даних. Метод:

* + Виконує SQL-запит для видалення запису з таблиці Tasks, використовуючи ідентифікатор TaskID.

1. Метод GetTasksByProjectId

public static List<Task> GetTasksByProjectId(int projectId)

Статичний метод повертає список завдань, пов’язаних із конкретним проектом. Він:

* + Виконує SQL-запит для отримання всіх записів із таблиці Tasks, що відповідають значенню ProjectID.
  + Створює об’єкти Task для кожного запису в базі даних і додає їх до списку.

1. Метод ToString

public override string ToString()

Повертає назву завдання у вигляді рядка, що корисно для виведення списків завдань у текстовому форматі.

Робота з базою даних

Клас Task взаємодіє з таблицею Tasks, яка має наступну структуру:

* TaskID — унікальний ідентифікатор завдання.
* ProjectID — зовнішній ключ, що вказує на проект, до якого належить завдання.
* TaskName — назва завдання.

Використання зовнішнього ключа ProjectID дозволяє підтримувати ієрархічну структуру даних, у якій одне завдання чітко прив’язане до конкретного проекту.

***Використання класу***

* Збереження завдання: користувач створює об’єкт Task і викликає метод Save, передаючи ідентифікатор проекту. Це додає завдання до бази даних.
* Отримання завдань: метод GetTasksByProjectId дозволяє завантажувати всі завдання для конкретного проекту, забезпечуючи їхній вивід у вигляді списку.
* Видалення завдань: метод Delete використовується для видалення обраного завдання з бази даних.

***Код класу* Task:**

using MySql.Data.MySqlClient;

using System.Collections.Generic;

namespace ProjectTracker

{

public class Task

{

public int TaskID { get; internal set; }

public string TaskName { get; set; }

public Task(string taskName)

{

TaskName = taskName;

}

public void Save(int projectId)

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «INSERT INTO Tasks (ProjectID, TaskName) VALUES (@ProjectID, @TaskName)»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@ProjectID», projectId);

cmd.Parameters.AddWithValue(«@TaskName», TaskName);

cmd.ExecuteNonQuery();

TaskID = (int)cmd.LastInsertedId;

}

}

}

public void Delete()

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «DELETE FROM Tasks WHERE TaskID = @TaskID»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@TaskID», TaskID);

cmd.ExecuteNonQuery();

}

}

}

public static List<Task> GetTasksByProjectId(int projectId)

{

List<Task> tasks = new List<Task>();

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «SELECT \* FROM Tasks WHERE ProjectID = @ProjectID»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@ProjectID», projectId);

using (var reader = cmd.ExecuteReader())

{

while (reader.Read())

{

Task = new Task(reader.GetString(«TaskName»))

{

TaskID = reader.GetInt32(«TaskID»),

};

tasks.Add(task);

}

}

}

}

return tasks;

}

public override string ToString()

{

return TaskName;

}

}

}

Клас Task є важливим компонентом додатку, що забезпечує управління завданнями в рамках проекту. Його інтеграція з базою даних дозволяє користувачам гнучко працювати із завданнями, зберігати їхній стан і динамічно оновлювати список завдань залежно від дій користувача.

***3.4.Опис розробки класу TimeEntry***

Клас TimeEntry є ключовим елементом для відстеження та зберігання інформації про витрачений час на виконання завдань у рамках проектів. Він дозволяє реєструвати часові записи для кожного проекту, включаючи час початку та завершення роботи, тривалість, і дату.

Основні властивості класу

* TimeEntryID *(тип: int)*: Унікальний ідентифікатор запису часу, який автоматично генерується базою даних після збереження.
* UserID *(тип: int)*: Ідентифікатор користувача, який створив часовий запис. Це дозволяє відстежувати, який користувач додав дані.
* StartTime *(тип: DateTime)*: Час початку роботи над проектом або завданням.
* EndTime *(тип: DateTime)*: Час завершення роботи.
* Duration *(тип: int, обчислювана властивість)*: Тривалість роботи в секундах, яка розраховується як різниця між EndTime і StartTime.
* ProjectID *(тип: int)*: Ідентифікатор проекту, до якого належить запис.
* EntryDate *(тип: DateTime)*: Дата створення запису, яка автоматично визначається на основі StartTime.

***Основний функціонал класу***

1. Конструктор
   * TimeEntry(int userId, int projectId, DateTime startTime, DateTime endTime)  
     Конструктор створює новий об’єкт часової реєстрації, задаючи необхідні параметри:
     + Ідентифікатор користувача.
     + Ідентифікатор проекту.
     + Час початку та завершення роботи.
     + Дату запису, яка автоматично визначається як дата StartTime.
2. Метод Save

public void Save()

Метод відповідає за збереження даних про часовий запис у базу даних:

* + Відкриває з’єднання з базою даних.
  + Виконує SQL-запит для додавання запису до таблиці timeentries, зберігаючи такі дані: UserID, StartTime, EndTime, Duration, ProjectID, та EntryDate.
  + Зберігає згенерований TimeEntryID у відповідну властивість.

***Робота з базою даних***

Клас TimeEntry інтегрується з таблицею timeentries, яка має наступну структуру:

* TimeEntryID — унікальний ідентифікатор запису.
* UserID — зовнішній ключ, що вказує на користувача.
* StartTime — час початку роботи.
* EndTime — час завершення роботи.
* Duration — тривалість у секундах.
* ProjectID — зовнішній ключ, що вказує на проект.
* EntryDate — дата запису.

Зовнішні ключі UserID і ProjectID забезпечують цілісність даних, дозволяючи однозначно пов’язувати часові записи з відповідними користувачами та проектами.

Використання класу

1. Створення часової реєстрації: користувач створює об’єкт TimeEntry, задаючи користувача, проект і часові межі роботи, після чого викликає метод Save, щоб зберегти дані в базу.
2. Аналіз активності: дані, збережені в об’єктах TimeEntry, можуть використовуватися для генерації статистики, зокрема аналізу часу, витраченого на виконання проектів.

***Код класу* TimeEntry:**

using MySql.Data.MySqlClient;

using System;

namespace ProjectTracker

{

public class TimeEntry

{

public int TimeEntryID { get; private set; }

public int UserID { get; set; }

public DateTime StartTime { get; set; }

public DateTime EndTime { get; set; }

public int Duration => (int)(EndTime – StartTime).TotalSeconds;

public int ProjectID { get; set; }

public DateTime EntryDate { get; set; }

public TimeEntry(int userId, int projectId, DateTime startTime, DateTime endTime)

{

UserID = userId;

ProjectID = projectId;

StartTime = startTime;

EndTime = endTime;

EntryDate = StartTime.Date;

}

public void Save()

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = «INSERT INTO timeentries (UserID, StartTime, EndTime, Duration, ProjectID, EntryDate) « +

«VALUES (@UserID, @StartTime, @EndTime, @Duration, @ProjectID, @EntryDate)»;

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue(«@UserID», UserID);

cmd.Parameters.AddWithValue(«@StartTime», StartTime);

cmd.Parameters.AddWithValue(«@EndTime», EndTime);

cmd.Parameters.AddWithValue(«@Duration», Duration);

cmd.Parameters.AddWithValue(«@ProjectID», ProjectID);

cmd.Parameters.AddWithValue(«@EntryDate», EntryDate);

cmd.ExecuteNonQuery();

TimeEntryID = (int)cmd.LastInsertedId;

}

}

}

}

}

Клас TimeEntry відіграє важливу роль у додатку, забезпечуючи функціонал для відстеження часу, витраченого користувачами на проекти. Його інтеграція з базою даних дозволяє зберігати та аналізувати часові дані, підтримуючи ефективне управління проектами та звітність.

***3.5.Опис розробки класу Database***

Клас Database виконує важливу роль у забезпеченні взаємодії додатку з базою даних taskmanagement. Він слугує єдиною точкою доступу для створення та управління з’єднаннями з базою даних, що дозволяє спростити код у всьому додатку та забезпечити централізоване управління підключеннями.

***Основна функціональність класу***

Клас Database містить такі функціональні елементи:

1. Поле connectionString

private string connectionString = «server=localhost;database=taskmanagement;user=Hb\_2908;password=4347#hB;»;

Це приватне поле визначає параметри підключення до бази даних, які включають:

* + server=localhost — адреса сервера бази даних. У цьому випадку це локальний сервер.
  + database=taskmanagement — ім’я бази даних, до якої здійснюється підключення.
  + user=Hb\_2908 — ім’я користувача бази даних.
  + password=4347#hB — пароль для доступу до бази даних.

Поле connectionString може бути змінене для підтримки інших серверів або параметрів доступу, що робить клас універсальним для різних середовищ.

1. Метод GetConnection

public MySqlConnection GetConnection()

Метод GetConnection створює новий об’єкт з’єднання з базою даних типу MySqlConnection. Його функціональність:

* + Використання рядка підключення connectionString для ініціалізації з’єднання.
  + Повернення нового екземпляра з’єднання, готового для використання в інших класах.

Завдяки цьому методу будь-який клас або компонент додатку може отримати доступ до бази даних, не турбуючись про параметри підключення.

***Взаємодія з базою даних***

Клас Database безпосередньо пов’язаний з базою даних taskmanagement, яка використовується для збереження та управління даними додатку. Він дозволяє:

* Підключатися до бази даних за необхідності.
* Забезпечити універсальність та спрощення доступу до бази для всіх компонентів.

***Використання класу***

1. Отримання з’єднання: Інші класи, такі як User, Project, Task, та TimeEntry, використовують метод GetConnection для створення та відкриття з’єднання з базою даних:

using (var connection = new Database().GetConnection())

{

connection.Open();

// Виконання запитів до бази даних

}

Завдяки цьому методи CRUD (Create, Read, Update, Delete) в кожному класі забезпечують надійний доступ до даних.

1. Централізоване управління з’єднаннями: Всі підключення здійснюються через єдину точку доступу, що знижує ризик помилок, пов’язаних із помилковими параметрами підключення.

***Переваги реалізації***

1. Інкапсуляція:  
   Поле connectionString приховане від зовнішнього доступу, що гарантує безпеку і недоступність параметрів з’єднання для сторонніх користувачів.
2. Гнучкість:  
   Якщо потрібно змінити сервер або параметри доступу, достатньо відредагувати лише одне поле.
3. Уніфікація:  
   Усі класи додатку використовують єдиний метод для створення підключень, що підвищує читабельність і спрощує підтримку коду.

***Код класу* Database:**

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ProjectTracker

{

public class Database

{

private string connectionString = «server=localhost;database=taskmanagement;user=Hb\_2908;password=4347#hB;»;

public MySqlConnection GetConnection()

{

return new MySqlConnection(connectionString);

}

}

}

Клас Database є фундаментальним компонентом додатку Project Tracker, забезпечуючи універсальний механізм підключення до бази даних taskmanagement. Завдяки цій реалізації знижується складність роботи з базою даних у інших класах, що підвищує ефективність розробки та спрощує майбутнє масштабування додатку.

**4. Розробка інтерфейсу та реалізація реєстрації і входу**

Для оформлення інтерфейсу було обрано бібліотеку ***MaterialSkin***, яка базується на концепції матеріального дизайну, розробленій Google. Ця бібліотека надає зручний набір інструментів для створення сучасного, естетичного, та інтуїтивно зрозумілого дизайну. Вибір ***MaterialSkin*** зумовлений кількома ключовими факторами. По-перше, бібліотека забезпечує можливість легкої інтеграції в Windows Forms додатки. По-друге, вона пропонує готові стилізовані елементи, які відповідають принципам матеріального дизайну, таким як тіні, плаваючі кнопки, чітка типографіка, та яскраві акцентні кольори.

***4.1. Особливості бібліотеки MaterialSkin***

Бібліотека ***MaterialSkin*** підтримує світлі й темні теми, що дозволяє адаптувати інтерфейс відповідно до потреб користувачів. Також вона має зручний механізм кастомізації кольорових схем, що дає змогу розробникам налаштовувати колірну палітру відповідно до загального стилю додатку. У цьому проекті було використано світлу тему з акцентами сірого та блакитного кольорів, що забезпечує приємний і професійний вигляд. Усі елементи — кнопки, текстові поля, посилання — виконані в єдиному стилі, що сприяє створенню цілісного враження від додатку.

***4.2. Реалізація форми реєстрації та входу***

Форма реєстрації та входу є однією з найважливіших частин додатку, оскільки вона забезпечує доступ користувачів до функціоналу. Основною метою було створити зручну та зрозумілу форму, яка б дозволила легко виконувати реєстрацію нових користувачів та авторизацію існуючих.

1. **Елементи форми** (рис.4.1)
   * **Поле для введення імені користувача**: Це поле з’являється лише під час реєстрації. Воно дозволяє користувачеві вказати унікальне ім’я, яке буде використано для персоналізації інтерфейсу.
   * **Поле для введення електронної пошти**: Це обов’язковий елемент як для реєстрації, так і для входу. Воно призначене для ідентифікації користувачів.
   * **Поле для введення пароля**: Для забезпечення конфіденційності даних у цьому полі використовується символ маскування, який приховує введений текст.
   * **Кнопки для реєстрації та входу**: Кожна з цих кнопок відповідає за виконання відповідної дії — реєстрації нового користувача або входу для існуючого.
   * **Посилання для переходу до реєстрації**: Це посилання дозволяє перемикатися між режимами реєстрації та входу.

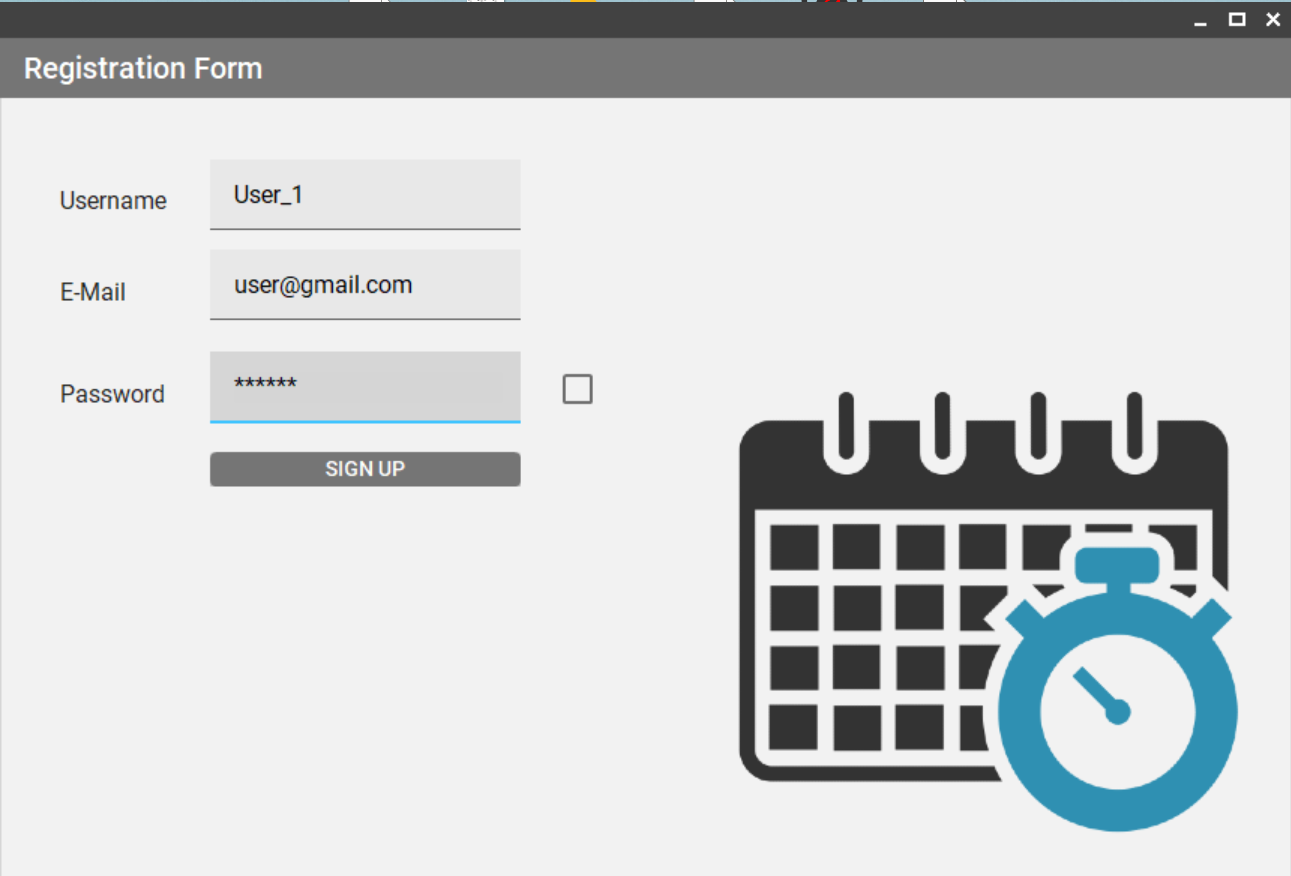


Рисунок 4.1 – Форма в режимі реєстрації

1. **Валідація введених даних** Усі поля форми мають вбудовані перевірки валідності. Наприклад, для електронної пошти перевіряється правильність формату, а для пароля — наявність мінімальної кількості символів. Якщо дані введені некоректно, користувач отримує відповідне повідомлення, що допомагає швидко виправити помилки.
2. **Механізм перемикання між режимами** Форма динамічно змінюється залежно від режиму. У режимі реєстрації з’являються додаткові поля для введення імені користувача, а також кнопка для завершення реєстрації. У режимі входу доступні лише поля для електронної пошти та пароля, а також кнопка для авторизації (рис.4.2).

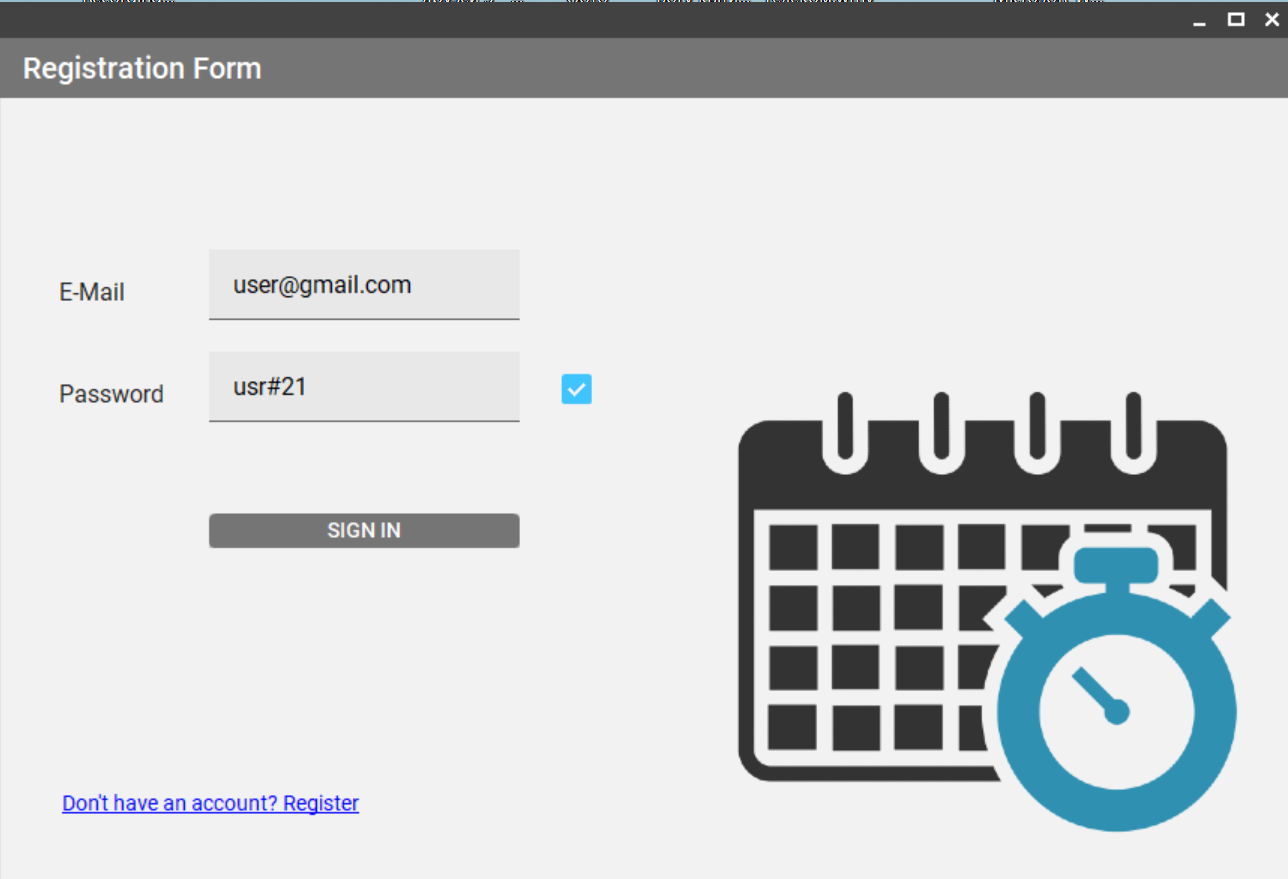


Рисунок 4.2 – Форма в режимі входу (пароль відображено)

**Додаткові можливості**

Форма також містить механізм для приховування або відображення пароля, що полегшує введення складних комбінацій. Реалізовано автоматичне очищення всіх полів після успішної реєстрації чи входу, що робить процес роботи з формою більш зручним.

***4.3.* *Код форми реєстрації та входу***

using MaterialSkin;

using MaterialSkin.Controls;

using System;

using System.Windows.Forms;

namespace ProjectTracker

{

public partial class RegistrationForm : MaterialForm

{

private bool isRegisterMode = false;

public RegistrationForm()

{

InitializeComponent();

InitializeMaterialDesign();

passwordTextBox.PasswordChar = ‘\*’;

ToggleMode();

}

private void InitializeMaterialDesign()

{

var materialSkinManager = MaterialSkinManager.Instance;

materialSkinManager.AddFormToManage(this);

materialSkinManager.Theme = MaterialSkinManager.Themes.LIGHT;

materialSkinManager.ColorScheme = new ColorScheme(

Primary.Grey600,

Primary.Grey800,

Primary.Grey200,

Accent.LightBlue200,

TextShade.WHITE

);

}

private void ToggleMode()

{

userNameTextBox.Visible = isRegisterMode;

userNameLabel.Visible = isRegisterMode;

signUpButton.Visible = isRegisterMode;

signInButton.Visible = !isRegisterMode;

registerLink.Visible = !isRegisterMode;

}

private void signUpButton\_Click(object sender, EventArgs e)

{

string username = userNameTextBox.Text;

string email = eMailTextBox.Text;

string password = passwordTextBox.Text;

string passwordHash = HashPassword(password);

if (User.FindByEmail(email) != null)

{

MessageBox.Show(«A user with this email already exists!», «Error», MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

User newUser = new User(username, email, passwordHash);

newUser.Save();

ClearFields();

ProjectsForm = new ProjectsForm(newUser.UserID, newUser.Username);

projectsForm.Show();

this.Hide();

}

private void signInButton\_Click(object sender, EventArgs e)

{

string email = eMailTextBox.Text;

string password = passwordTextBox.Text;

User existingUser = User.FindByEmail(email);

if (existingUser == null)

{

MessageBox.Show(«User with this email does not exist!», «Error», MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (VerifyPassword(password, existingUser.PasswordHash))

{

ClearFields();

ProjectsForm = new ProjectsForm(existingUser.UserID, existingUser.Username);

projectsForm.Show();

this.Hide();

}

else

{

MessageBox.Show(«Incorrect password!», «Error», MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void registerLink\_LinkClicked\_1(object sender, LinkLabelLinkClickedEventArgs e)

{

isRegisterMode = true;

ToggleMode();

}

private void ClearFields()

{

userNameTextBox.Text = «»;

eMailTextBox.Text = «»;

passwordTextBox.Text = «»;

}

private string HashPassword(string password)

{

return BCrypt.Net.BCrypt.HashPassword(password);

}

private bool VerifyPassword(string enteredPassword, string storedHash)

{

return BCrypt.Net.BCrypt.Verify(enteredPassword, storedHash);

}

private void togglePasswordVisibility\_CheckedChanged(object sender, EventArgs e)

{

passwordTextBox.PasswordChar = togglePasswordVisibility.Checked ? ‘\0’ : ‘\*’;

}

}

}

Результатом стало створення зручного, стильного інтерфейсу , який відповідає сучасним стандартам, забезпечуючи як естетичну привабливість, так і функціональну ефективність.

**5. Розробка основної сторінки з проектами і таймером**

Основна сторінка додатку була розроблена для забезпечення зручного та інтуїтивно зрозумілого інтерфейсу управління проектами користувача. Вона виконує функцію основного робочого простору, де користувач може бачити всі свої проекти, створювати нові, редагувати існуючі, додавати та переглядати завдання, а також запускати та зупиняти таймер для відстеження часу роботи над кожним проектом.

***5.1.Структура сторінки***

Основна сторінка включає кілька ключових елементів, кожен із яких виконує певну роль у взаємодії користувача з проектами (рис.5.1):

1. **Заголовок та привітання**: У верхній частині розміщено текстовий елемент, який показує ім’я користувача. Це надає сторінці персоналізованості та підтверджує, що користувач успішно увійшов у систему.
2. **Кнопка додавання проекту**: Основний інструмент для створення нових проектів. При натисканні відкривається форма, де користувач вводить назву нового проекту. Після збереження дані про новий проект додаються до бази даних, а сам проект з’являється на сторінці.
3. **Картки проектів**: Кожен проект представлений у вигляді окремої картки. Картка містить інформацію про проект, включаючи назву, загальний час, витрачений на виконання, і список завдань, пов’язаних із проектом. Це дозволяє користувачеві швидко переглядати основну інформацію без необхідності відкривати додаткові форми.
4. **Елементи керування проектами**:
   * **Список завдань**: Відображається у вигляді випадаючого списку, де автоматично завантажуються всі завдання, пов’язані із поточним проектом. Якщо завдання є, перше з них вибирається за замовчуванням.
   * **Кнопки управління завданнями**: Користувач може додавати нові завдання до проекту або видаляти існуючі. Після додавання завдання список оновлюється автоматично.
   * **Таймер**: Для кожного проекту є можливість запуску і зупинки таймера. Таймер відображає години, хвилини і секунди, що минули від початку роботи. Коли таймер зупиняється, витрачений час додається до загального часу проекту, який також оновлюється на карточці.
   * **Кнопка видалення проекту**: Дозволяє видалити весь проект разом із пов’язаними завданнями. Після видалення карточка проекту зникає зі сторінки.
5. **Глобальна статистика**: У нижній частині сторінки передбачено кнопку для перегляду загальної статистики. Вона відкриває нове вікно, де відображаються підсумкові дані про всі проекти користувача.

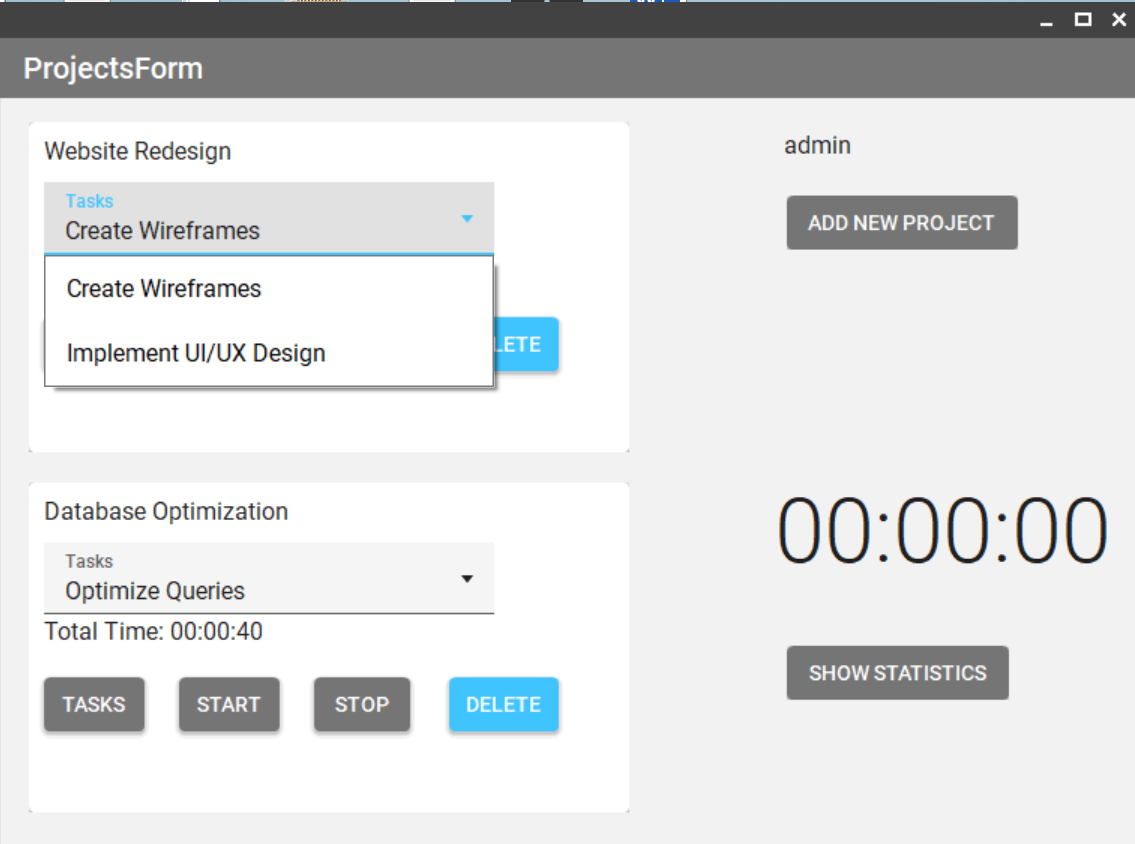


Рисунок 5.1 – Зовнішній вигляд основної сторінки додатку

***5.2.Функціональні можливості***

* **Додавання проектів**: Користувач має можливість створювати нові проекти через спеціальну форму. Після введення назви проект додається в базу даних, і його карточка автоматично з’являється на сторінці.
* **Керування завданнями**: Користувач може переглядати, додавати та видаляти завдання для кожного проекту. Завдання зберігаються в базі даних, і кожне з них пов’язане із відповідним проектом.
* **Таймер**: Таймер дозволяє відстежувати час, витрачений на роботу над проектом. Він починає відлік при запуску і зупиняється за командою користувача. Після зупинки витрачений час додається до загального часу проекту, а також записується у базу даних як окремий запис із деталями про час початку, завершення і тривалість.
* **Видалення проектів**: Видалення проекту передбачає також видалення всіх пов’язаних із ним завдань. Після видалення карточка проекту і відповідні дані видаляються з бази даних.

***5.3.Графічний дизайн***

Для дизайну сторінки також було використано бібліотеку ***MaterialSkin***. Це забезпечує сучасний і привабливий вигляд елементів інтерфейсу, таких як кнопки, випадаючі списки, текстові поля і заголовки. Завдяки адаптивному дизайну сторінка виглядає естетично як на великих екранах, так і на менших пристроях.

***MaterialSkin*** дозволяє задати світлу тему, яка підходить для професійного і інтуїтивного робочого середовища. Для кожного елемента були налаштовані кольорові схеми, які гармонійно поєднуються з темою додатку, з використанням акцентів іконок для дій, таких як запуск або зупинка таймера.

***5.4.Код сторінки ProjectsForm***

using MaterialSkin;

using MaterialSkin.Controls;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace ProjectTracker

{

public partial class ProjectsForm : MaterialForm

{

private int userId;

private List<Project> projects;

private int projectCardY = 80;

private int hours = 0, minutes = 0, seconds = 0;

private Project currentProject;

private DateTime? startTime;

public ProjectsForm(int userId, string username)

{

InitializeComponent();

InitializeMaterialDesign();

this.userId = userId;

projects = new List<Project>();

userLabel.Text = $»{username}»;

projectTimer.Tick += ProjectTimer\_Tick;

}

private void InitializeMaterialDesign()

{

var materialSkinManager = MaterialSkinManager.Instance;

materialSkinManager.AddFormToManage(this);

materialSkinManager.Theme = MaterialSkinManager.Themes.LIGHT;

materialSkinManager.ColorScheme = new ColorScheme(

Primary.Grey600,

Primary.Grey800,

Primary.Grey200,

Accent.LightBlue200,

TextShade.WHITE

);

}

private void addProjectButton\_Click(object sender, EventArgs e)

{

var addProjectForm = new AddProjectForm();

if (addProjectForm.ShowDialog() == DialogResult.OK)

{

Project newProject = new Project(userId, addProjectForm.ProjectName);

newProject.Save();

projects.Add(newProject);

DisplayProject(newProject);

}

}

private void DisplayProject(Project project)

{

MaterialCard projectCard = new MaterialCard

{

Width = 400,

Height = 220,

Margin = new Padding(10),

Padding = new Padding(10),

Location = new Point(20, projectCardY)

};

MaterialLabel projectNameLabel = new MaterialLabel

{

Text = project.ProjectName,

Font = new Font(«Roboto», 14, FontStyle.Bold),

Location = new Point(10, 10),

AutoSize = true

};

MaterialComboBox taskComboBox = new MaterialComboBox

{

Width = 300,

Location = new Point(10, 40),

Hint = «Tasks»

};

UpdateTaskComboBox(project, taskComboBox);

MaterialLabel projectTimeLabel = new MaterialLabel

{

Text = $»Total Time: {FormatTime(project.TotalTime)}»,

Location = new Point(10, 90),

AutoSize = true

};

MaterialButton addTaskButton = new MaterialButton

{

Text = «Tasks»,

Location = new Point(10, 130),

Width = 80

};

addTaskButton.Click += (s, e) => OpenAddTaskForm(project, taskComboBox);

MaterialButton startButton = new MaterialButton

{

Text = «Start»,

Location = new Point(100, 130),

Width = 80

};

startButton.Click += (s, e) => StartTimer(project);

MaterialButton stopButton = new MaterialButton

{

Text = «Stop»,

Location = new Point(190, 130),

Width = 80

};

stopButton.Click += (s, e) => StopTimer(project, projectTimeLabel);

MaterialButton deleteButton = new MaterialButton

{

Text = «Delete»,

Location = new Point(280, 130),

Width = 80,

HighEmphasis = true,

UseAccentColor = true

};

deleteButton.Click += (s, e) => DeleteProject(project, projectCard);

projectCard.Controls.Add(projectNameLabel);

projectCard.Controls.Add(taskComboBox);

projectCard.Controls.Add(projectTimeLabel);

projectCard.Controls.Add(addTaskButton);

projectCard.Controls.Add(startButton);

projectCard.Controls.Add(stopButton);

projectCard.Controls.Add(deleteButton);

Controls.Add(projectCard);

projectCardY += projectCard.Height + 20;

}

private void OpenAddTaskForm(Project project, MaterialComboBox taskComboBox)

{

var addTaskForm = new AddTaskForm(project.ProjectID);

var result = addTaskForm.ShowDialog();

if (result == DialogResult.OK || result == DialogResult.Cancel)

{

project.LoadTasks();

UpdateTaskComboBox(project, taskComboBox);

}

}

private void UpdateTaskComboBox(Project project, MaterialComboBox taskComboBox)

{

taskComboBox.Items.Clear();

project.LoadTasks();

foreach (var task in project.Tasks)

{

taskComboBox.Items.Add(task);

}

if (taskComboBox.Items.Count > 0)

{

taskComboBox.SelectedIndex = 0;

}

}

private void StartTimer(Project project)

{

currentProject = project;

startTime = DateTime.Now;

hours = minutes = seconds = 0;

projectTimer.Start();

}

private void StopTimer(Project project, MaterialLabel projectTimeLabel)

{

projectTimer.Stop();

DateTime endTime = DateTime.Now;

int elapsedTime = hours \* 3600 + minutes \* 60 + seconds;

project.UpdateTotalTime(elapsedTime);

projectTimeLabel.Text = $»Total Time: {FormatTime(project.TotalTime)}»;

if (startTime.HasValue)

{

TimeEntry = new TimeEntry(userId, project.ProjectID, startTime.Value, endTime);

timeEntry.Save();

}

timerDisplayLabel.Text = «00:00:00»;

startTime = null;

currentProject = null;

}

private void ProjectTimer\_Tick(object sender, EventArgs e)

{

seconds++;

if (seconds >= 60)

{

seconds = 0;

minutes++;

}

if (minutes >= 60)

{

minutes = 0;

hours++;

}

timerDisplayLabel.Text = $»{hours:D2}:{minutes:D2}:{seconds:D2}»;

}

private string FormatTime(int totalSeconds)

{

int hours = totalSeconds / 3600;

int minutes = (totalSeconds % 3600) / 60;

int seconds = totalSeconds % 60;

return $»{hours:D2}:{minutes:D2}:{seconds:D2}»;

}

private void DeleteProject(Project project, MaterialCard projectCard)

{

project.Delete();

Controls.Remove(projectCard);

projects.Remove(project);

UpdateProjectCardPositions();

}

private void showStatisticsButton\_Click(object sender, EventArgs e)

{

var statisticsForm = new StatisticsForm(userId);

statisticsForm.Show();

}

private void UpdateProjectCardPositions()

{

projectCardY = 80;

foreach (Control in Controls)

{

if (control is MaterialCard)

{

control.Location = new Point(20, projectCardY);

projectCardY += control.Height + 20;

}

}

}

private void ProjectsForm\_Load(object sender, EventArgs e)

{

projects = Project.GetProjectsByUser(userId);

foreach (var project in projects)

{

DisplayProject(project);

}

}

}

}

Основна сторінка додатку реалізована з урахуванням зручності користувача та забезпечує всі необхідні функції для управління проектами. Розроблений інтерфейс інтуїтивний, а функціональність таймера і можливість детального перегляду статистики підвищують ефективність роботи з додатком.

**6. Розробка форми для додавання проектів**

Форма для додавання проектів забезпечує простий і зручний інтерфейс, що дозволяє користувачам створювати нові проекти. Її головна мета — забезпечити можливість введення назви проекту та передачі цієї інформації для подальшого збереження у базі даних. У процесі розробки було враховано естетичність, інтуїтивність та відповідність загальному стилю програми.

Для візуального оформлення форми було обрано бібліотеку **MaterialSkin**, яка дозволяє створювати інтерфейси у сучасному та привабливому стилі. Ця бібліотека забезпечує консистентний вигляд програми завдяки використанню стандартних кольорових схем, типографії та інтерактивних елементів, характерних для дизайну Material Design. У рамках налаштування форми було обрано світлу тему з акцентом на нейтральні відтінки сірого та світло-блакитний колір для інтерактивних елементів.

Форма містить такі основні елементи (рис.6.1):

1. **Текстове поле для введення назви проекту.**

Цей елемент дозволяє користувачам вводити текстову інформацію. Було передбачено перевірку на порожнє значення, щоб уникнути створення проектів без назви. Якщо користувач не ввів жодного тексту, з’являється повідомлення про необхідність заповнення поля.

1. **Кнопка підтвердження.**

Ця кнопка виконує функцію підтвердження введеної інформації. Після натискання відбувається перевірка введених даних. Якщо назва проекту заповнена коректно, форма закривається, а введена інформація передається для подальшої обробки.

1. **Кольорова схема та стиль.**

Усі елементи форми гармонійно інтегровані у загальний дизайн програми завдяки використанню бібліотеки MaterialSkin. Тема світла, що забезпечує приємний для очей вигляд, а кольорові акценти сприяють легкому розпізнаванню інтерактивних елементів.

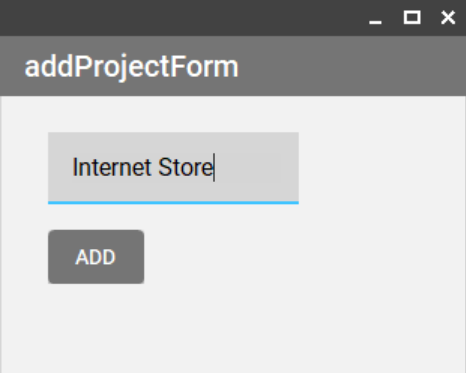


Рисунок 6.1 – Форма для додавання проекту

Форма була розроблена таким чином, щоб забезпечити інтуїтивне використання та мінімум дій для завершення завдання. Застосовані валідаційні механізми дозволяють уникнути помилок у процесі введення даних, а зрозумілий інтерфейс спрощує процес додавання нових проектів.

***6.1.Код форми для додавання проекту***

using MaterialSkin.Controls;

using MaterialSkin;

using System;

using System.Windows.Forms;

namespace ProjectTracker

{

public partial class AddProjectForm : MaterialForm

{

public string ProjectName { get; private set; }

public AddProjectForm()

{

InitializeComponent();

InitializeMaterialDesign();

}

private void InitializeMaterialDesign()

{

var materialSkinManager = MaterialSkinManager.Instance;

materialSkinManager.AddFormToManage(this);

materialSkinManager.Theme = MaterialSkinManager.Themes.LIGHT;

materialSkinManager.ColorScheme = new ColorScheme(

Primary.Grey600,

Primary.Grey800,

Primary.Grey200,

Accent.LightBlue200,

TextShade.WHITE

);

}

private void addButton\_Click(object sender, EventArgs e)

{

ProjectName = projectNameTextBox.Text;

if (string.IsNullOrWhiteSpace(ProjectName))

{

MessageBox.Show("Please enter a project name.");

return;

}

DialogResult = DialogResult.OK;

Close();

}

}

}

Ця форма є важливим компонентом програми, оскільки саме з неї починається управління новими проектами. Її продумана структура та сучасний вигляд підвищують комфорт роботи користувача та відповідають загальним принципам розробки якісного програмного забезпечення.

**7. Розробка форми управління завданнями**

Форма управління завданнями була створена для забезпечення ефективного керування завданнями, які входять до складу окремого проекту. Ця форма дозволяє користувачам додавати нові завдання, переглядати вже існуючі, та видаляти непотрібні завдання. Реалізація функціоналу була виконана з урахуванням потреб користувача у швидкому та зрозумілому інтерфейсі.

***7.1.Загальний опис функціональності***

Дана форма відіграє ключову роль у деталізації проектів, дозволяючи користувачам розділяти великі задачі на окремі завдання. Завдяки цій формі користувачі можуть:

* **Додавати нові завдання** із зазначенням їхніх назв.
* **Переглядати список існуючих завдань**, які належать до вибраного проекту.
* **Видаляти завдання**, які більше не є актуальними.

***7.2.*Особливості оформлення**

Як і інші форми у додатку, ця форма була створена з використанням бібліотеки **MaterialSkin**, що забезпечує стильний та сучасний вигляд. Бібліотека дозволяє створювати компоненти інтерфейсу у стилі Material Design з готовими шаблонами кольорів, шрифтів та анімацій. Для цієї форми була обрана світла тема з використанням сірого та блакитного акцентів.

***7.3.Основні елементи форми***

1. **Поле введення для назви завдання.**

Це текстове поле дозволяє користувачам ввести назву нового завдання, яке потрібно додати до проекту. Було передбачено валідацію введення: якщо поле залишається порожнім, користувач отримує сповіщення про необхідність вказати назву.

1. **Випадаючий список для перегляду завдань.**

У цьому списку відображаються всі завдання, які вже були створені для обраного проекту. Список оновлюється у реальному часі після додавання або видалення завдання.

1. **Кнопка для додавання завдання.**

Натискання цієї кнопки активує процес створення нового завдання. Назва завдання береться із текстового поля, а саме завдання додається до бази даних та списку завдань у формі. Після додавання текстове поле очищається для зручності користувача.

1. **Кнопка для видалення завдання.**

Ця кнопка дозволяє видалити вибране завдання з бази даних та зі списку завдань. У випадку, якщо завдання не було обране, користувач отримує відповідне сповіщення.

1. **Кольорова схема та дизайн.**

Усі компоненти форми інтегровані у загальну стилістику програми. Випадаючий список та кнопки мають інтерактивні ефекти, які покращують досвід роботи з додатком.

***7.4.Процес взаємодії користувача з формою***

Користувачі можуть почати з додавання нових завдань, вводячи їхні назви у текстове поле та натискаючи кнопку підтвердження (рис.7.1). Список завдань автоматично оновлюється, відображаючи додані елементи (рис.7.2). Для видалення завдання достатньо вибрати його у списку та натиснути кнопку видалення. Усі дії виконуються швидко та ефективно, з урахуванням інтерактивних підказок та сповіщень.

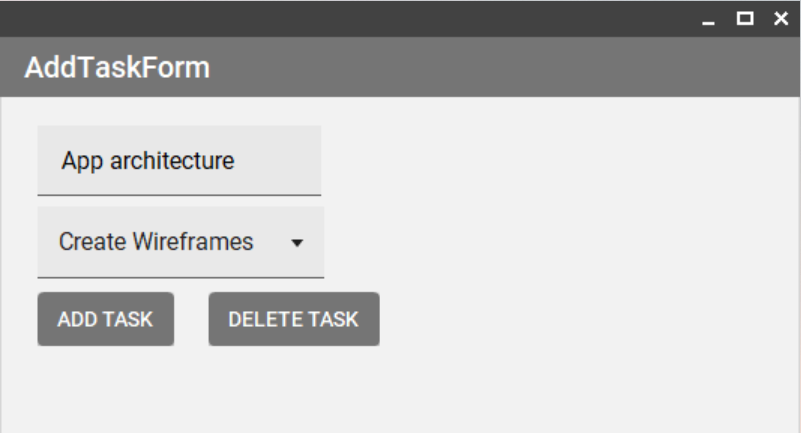


Рисунок 7.1 – Ввід у форму назви нового завдання

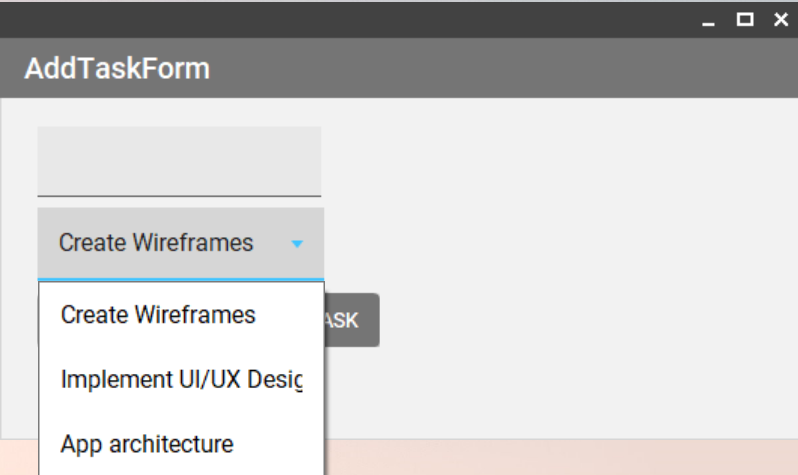


Рисунок 7.2 – Завдання успішно додано до списку

***7.5.Код форми управління проектами***

using MaterialSkin;

using MaterialSkin.Controls;

using System;

using System.Windows.Forms;

namespace ProjectTracker

{

public partial class AddTaskForm : MaterialForm

{

public Task CreatedTask { get; private set; }

private int projectId;

public AddTaskForm(int projectId)

{

InitializeComponent();

this.projectId = projectId;

InitializeMaterialDesign();

LoadTasks();

}

private void InitializeMaterialDesign()

{

var materialSkinManager = MaterialSkinManager.Instance;

materialSkinManager.AddFormToManage(this);

materialSkinManager.Theme = MaterialSkinManager.Themes.LIGHT;

materialSkinManager.ColorScheme = new ColorScheme(

Primary.Grey600,

Primary.Grey800,

Primary.Grey200,

Accent.LightBlue200,

TextShade.WHITE

);

}

private void LoadTasks()

{

taskComboBox.Items.Clear();

var tasks = Task.GetTasksByProjectId(projectId);

foreach (var task in tasks)

{

taskComboBox.Items.Add(task);

}

if (taskComboBox.Items.Count > 0)

{

taskComboBox.SelectedIndex = 0;

}

}

private void addTaskButton\_Click(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(taskNameTextBox.Text))

{

CreatedTask = new Task(taskNameTextBox.Text);

CreatedTask.Save(projectId);

taskComboBox.Items.Add(CreatedTask);

taskNameTextBox.Clear();

}

else

{

MessageBox.Show("Please enter a task name.");

}

}

private void deleteTaskButton\_Click(object sender, EventArgs e)

{

if (taskComboBox.SelectedItem is Task selectedTask)

{

selectedTask.Delete();

taskComboBox.Items.Remove(selectedTask);

}

else

{

MessageBox.Show("Please select a task to delete.");

}

}

}

}

Форма управління завданнями є важливим компонентом програми, оскільки вона дозволяє структурувати проекти, забезпечуючи деталізацію робочого процесу. Її функціональність сприяє покращенню організації роботи, а продуманий інтерфейс робить взаємодію користувача максимально комфортною.

**8. Розробка форми для перегляду статистики**

Форма для перегляду статистики була створена з метою забезпечення користувачів можливістю аналізувати свою продуктивність у рамках роботи над проектами. Вона дозволяє переглядати загальний час, витрачений на різні проекти, за певні часові періоди. Інтерфейс форми забезпечує інтуїтивно зрозумілу взаємодію, а результати представлені у наочному графічному вигляді, що дозволяє швидко оцінити прогрес.

***8.1.Загальний опис функціональності***

Дана форма забезпечує:

* **Перегляд статистики за обраний період.** Користувач може обрати перегляд даних за тиждень або за місяць.
* **Візуалізацію даних.** Витрачений час на проекти відображається у вигляді діаграми, що спрощує аналіз.
* **Динамічне оновлення даних.** Графік змінюється в реальному часі залежно від вибору періоду або нових записів у базі даних.

***8.2.Особливості оформлення***

Форма для перегляду статистики була створена з використанням бібліотеки **MaterialSkin**, яка забезпечує стильний та сучасний дизайн. Усі компоненти форми гармонійно вписуються у загальну стилістику програми завдяки використанню світлої теми та акцентних кольорів. Це дозволяє користувачам комфортно працювати з формою незалежно від часу та умов освітлення.

Для візуалізації даних використано стандартний елемент **Chart** із бібліотеки Windows Forms DataVisualization. Ця бібліотека надає можливість створювати діаграми різних типів, таких як стовпчикові, кругові чи лінійні, забезпечуючи високу інтерактивність і налаштовуваність. У даному проекті було обрано стовпчикову діаграму, яка оптимально підходить для порівняння витраченого часу між проектами.

**8.3.Основні елементи форми**

1. **Елементи для вибору періоду.** На формі передбачені перемикачі, які дозволяють користувачу обрати період для перегляду статистики — тиждень або місяць. При виборі одного з варіантів дані автоматично оновлюються.
2. **Діаграма для відображення даних.** Основна частина форми відведена під інтерактивну діаграму, яка відображає назви проектів та загальний час, витрачений на кожен із них. Це дозволяє візуально оцінити внесок у різні проекти за обраний період.
3. **Динамічне оновлення графіка.** Після вибору періоду відбувається запит до бази даних, і результати одразу відображаються на діаграмі. Це робить процес перегляду статистики швидким та зручним.

***8.4.Процес взаємодії з формою***

Користувач починає роботу з формою, обираючи бажаний період для перегляду статистики (рис.8.1). Після вибору формується запит до бази даних, який збирає дані про проекти та загальний час, витрачений на них за відповідний період. Ці дані виводяться на діаграму, яка автоматично оновлюється.

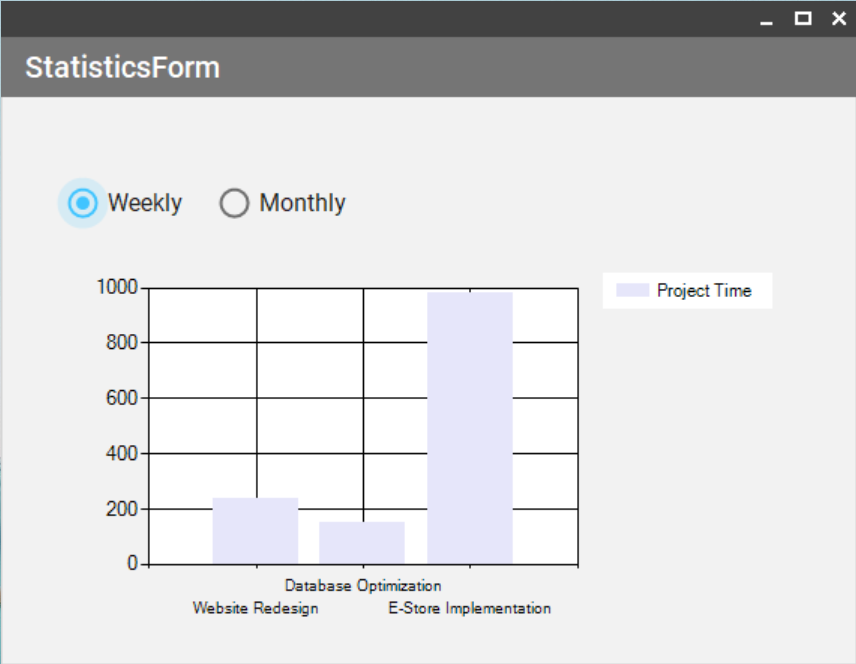


Рисунок 8.1 – Статистика часу, витраченого на кожен з проектів

Всі дані у формі подаються у структурованому та наочному вигляді, що забезпечує простоту аналізу та прийняття рішень. Інтерактивність форми, зокрема можливість швидкого перемикання між періодами, робить її важливим інструментом для оцінки продуктивності.

***8.5.Код форми для перегляду статистики***

using MaterialSkin;

using MaterialSkin.Controls;

using MySql.Data.MySqlClient;

using System;

using System.Data;

namespace ProjectTracker

{

public partial class StatisticsForm : MaterialForm

{

private int userId;

public StatisticsForm(int userId)

{

InitializeComponent();

InitializeMaterialDesign();

this.userId = userId;

}

private void InitializeMaterialDesign()

{

var materialSkinManager = MaterialSkinManager.Instance;

materialSkinManager.AddFormToManage(this);

materialSkinManager.Theme = MaterialSkinManager.Themes.LIGHT;

materialSkinManager.ColorScheme = new ColorScheme(

Primary.Grey600,

Primary.Grey800,

Primary.Grey200,

Accent.LightBlue200,

TextShade.WHITE

);

}

private void weeklyRadioButton\_CheckedChanged(object sender, EventArgs e)

{

LoadStatistics("weekly");

}

private void monthlyRadioButton\_CheckedChanged(object sender, EventArgs e)

{

LoadStatistics("monthly");

}

private void LoadStatistics(string period)

{

using (var connection = new Database().GetConnection())

{

connection.Open();

string query = period == "weekly"

? @"SELECT ProjectName, SUM(Duration) AS TotalTime

FROM timeentries

JOIN projects ON timeentries.ProjectID = projects.ProjectID

WHERE timeentries.UserID = @UserID AND EntryDate >= DATE\_SUB(CURDATE(), INTERVAL 1 WEEK)

GROUP BY ProjectName"

: @"SELECT ProjectName, SUM(Duration) AS TotalTime

FROM timeentries

JOIN projects ON timeentries.ProjectID = projects.ProjectID

WHERE timeentries.UserID = @UserID AND EntryDate >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)

GROUP BY ProjectName";

using (var cmd = new MySqlCommand(query, connection))

{

cmd.Parameters.AddWithValue("@UserID", userId);

var dataTable = new DataTable();

var adapter = new MySqlDataAdapter(cmd);

adapter.Fill(dataTable);

DisplayChart(dataTable);

}

}

}

private void DisplayChart(DataTable dataTable)

{

statisticsChart.Series["Project Time"].Points.Clear();

foreach (DataRow row in dataTable.Rows)

{

statisticsChart.Series["Project Time"].Points.AddXY(row["ProjectName"], row["TotalTime"]);

}

}

}

}

Форма для перегляду статистики є важливим елементом програми, який дозволяє користувачам отримувати повну картину своєї роботи над проектами. Її реалізація забезпечує ефективний контроль за часом, витраченим на різні задачі, та сприяє покращенню продуктивності.

**9. Висновки та перспективи розвитку**

У процесі розробки програмного забезпечення **"Project Tracker"** було реалізовано програмний продукт, який забезпечує зручний інструмент для управління проектами та завданнями. Цей додаток поєднує в собі сучасні програмні підходи, зрозумілий інтерфейс та функціональність, яка відповідає актуальним вимогам користувачів. Його розробка стала яскравим прикладом застосування об'єктно-орієнтованого програмування на мові **C#** із використанням технологій платформи **.NET Framework**.

***9.1.Використані технології та підходи***

Проект розроблявся з урахуванням принципів модульності, повторного використання коду та масштабованості. Застосування об'єктно-орієнтованого підходу дозволило структурувати функціональність додатку у вигляді окремих класів, таких як користувачі, проекти, завдання та записи часу. Кожен клас має чітко визначену відповідальність, що спрощує як початкову розробку, так і підтримку коду в майбутньому.

Для роботи з базою даних було обрано **MySQL** у поєднанні з бібліотекою **MySql.Data**, яка забезпечила зручний доступ до даних. Структура бази даних побудована таким чином, щоб уникнути дублювання інформації та зберігати її у зв’язному вигляді. Зокрема, таблиці пов’язані між собою через зовнішні ключі, що забезпечує цілісність даних.

Особливу увагу було приділено створенню користувацького інтерфейсу, для якого використано бібліотеку **MaterialSkin**. Ця бібліотека дозволила реалізувати сучасний дизайн із чіткими візуальними елементами, що забезпечують зрозумілий та інтуїтивний інтерфейс. Вибір MaterialSkin обумовлений її відповідністю сучасним тенденціям у розробці інтерфейсів та її сумісністю з платформою **Windows Forms**.

***9.2.Функціональні досягнення***

Додаток забезпечує користувачів необхідним функціоналом для управління проектами:

* Реалізовано механізм авторизації та реєстрації, який забезпечує безпеку облікових записів через використання хешування паролів.
* Створено інтерфейс для додавання, редагування та видалення проектів і завдань, що дозволяє користувачам легко налаштовувати свої робочі процеси.
* Впроваджено таймер, який дозволяє відстежувати витрачений час на проекти, що допомагає аналізувати ефективність виконаної роботи.
* Реалізовано можливість перегляду статистики у вигляді графіків, що візуалізують результати роботи за обраний період.

***9.3.Перспективи вдосконалення***

Хоча додаток "Project Tracker" вже є функціональним, існує низка напрямків для його вдосконалення. Насамперед можна додати логіку для детального відстеження часу, витраченого на кожне окреме завдання. Це дозволить користувачам отримувати більш глибокий аналіз своєї продуктивності, а також допоможе у плануванні роботи.

Ще одним перспективним напрямком є розширення можливостей статистичного аналізу. Наприклад, можна впровадити функціонал для виведення звітів у вигляді PDF-документів, що може бути корисним для презентації результатів роботи командам або керівникам.

З точки зору інтерфейсу, додаток можна вдосконалити шляхом впровадження адаптивного дизайну, який забезпечить коректне відображення на різних розмірах екранів. Крім того, можливість інтеграції з мобільними платформами дозволить розширити аудиторію користувачів.

Також можливе впровадження багатокористувацького режиму, що дозволить користувачам спільно працювати над проектами, делегувати завдання та стежити за прогресом у реальному часі. Для цього можна використати хмарні сервіси для синхронізації даних.

***9.4.Загальні висновки***

Розробка "Project Tracker" продемонструвала ефективність об’єктно-орієнтованого підходу (ООП) у створенні додатків для платформи Windows. Завдяки цьому підходу вдалося забезпечити чітку структурованість програми, де кожен компонент виконує конкретну роль і має свої межі відповідальності. Наприклад, класи для роботи з користувачами, проектами та завданнями містять усі необхідні методи для обробки даних та взаємодії з базою, що дозволяє ізолювати їхню логіку від інших частин програми. Це робить код більш гнучким і зрозумілим, а також сприяє зменшенню ймовірності помилок при його подальшій модифікації.

Інкапсуляція стала ключовою у створенні додатку, оскільки дозволила приховати внутрішні деталі реалізації класів та зробити взаємодію між компонентами більш безпечною. Повторне використання коду в рамках проекту також стало можливим завдяки ретельному проектуванню класів. Наприклад, методи для збереження чи отримання даних з бази були реалізовані таким чином, щоб їх можна було адаптувати до нових потреб без значних змін у логіці програми.

ООП також дозволяє легко масштабувати додаток. У випадку, якщо виникає потреба у додаванні нового функціоналу, це можна зробити без суттєвого впливу на існуючу архітектуру. Так, для розширення проекту, наприклад, шляхом впровадження функціоналу для запису часу, витраченого на конкретне завдання, достатньо буде додати кілька методів у відповідний клас, не зачіпаючи основну структуру.

Не менш важливу роль у реалізації проекту відіграла платформа .NET Framework. Її бібліотеки дозволили значно прискорити процес розробки, надаючи готові рішення для роботи з базою даних, створення графічного інтерфейсу та управління подіями. Використання бібліотеки MaterialSkin для оформлення інтерфейсу допомогло створити сучасний, естетично привабливий дизайн, який відповідає вимогам сучасних користувачів.

"Project Tracker" став демонстрацією ефективності сучасних технологій C# та .NET у створенні функціонального і зручного інструменту для управління проектами. Досягнуті результати підтверджують правильність обраного підходу, а можливість подальшого розвитку додатку відкриває нові перспективи для покращення його функціоналу та зручності використання.